

? show files;ds

File 347:JAPIO Dec 1976-2006/Dec(Updated 070403)

(c) 2007 JPO & JAPIO

File 348:EUROPEAN PATENTS 1978-2007/ 200708

(c) 2007 European Patent Office

File 349:PCT FULLTEXT 1979-2007/UB=20070329UT=20070322

(c) 2007 WIPO/Thomson

File 350:Derwent WPIX 1963-2006/UD=200722

(c) 2007 The Thomson Corporation

File 371:French Patents 1961-2002/BOPI 200209

(c) 2002 INPI. All rts. reserv.

File 120:U.S. Copyrights 1978-2007/Mar 20

(c) format only 2007 Dialog

File 426:LCMARC-Books 1968-2007/Apr W1

(c) format only 2007 Dialog

File 430:British Books in Print 2007/Jan W3

(c) 2007 J. Whitaker & Sons Ltd.

File 483:Newspaper Abs Daily 1986-2007/Apr 05

(c) 2007 ProQuest Info&Learning

File 2:INSPEC 1898-2007/Mar W4

(c) 2007 Institution of Electrical Engineers

File 35:Dissertation Abs Online 1861-2007/Mar

(c) 2007 ProQuest Info&Learning

File 65:Inside Conferences 1993-2007/Apr 05

(c) 2007 BLDSC all rts. reserv.

File 99:Wilson Appl. Sci & Tech Abs 1983-2007/Mar

(c) 2007 The HW Wilson Co.

File 474:New York Times Abs 1969-2007/Apr 04

(c) 2007 The New York Times

File 475:Wall Street Journal Abs 1973-2007/Apr 04

(c) 2007 The New York Times

File 583:Gale Group Globalbase(TM) 1986-2002/Dec 13

(c) 2002 The Gale Group

File 256:TecInfoSource 82-2007/Oct

(c) 2007 Info.Sources Inc

File 139:EconLit 1969-2007/Mar

(c) 2007 American Economic Association

File 6:NTIS 1964-2007/Apr W1

(c) 2007 NTIS, Intl Cpyrght All Rights Res

File 8:Ei Compendex(R) 1884-2007/Mar W4

(c) 2007 Elsevier Eng. Info. Inc.

File 34:SciSearch(R) Cited Ref Sci 1990-2007/Apr W1

(c) 2007 The Thomson Corp

File 144:Pascal 1973-2007/Mar W4

(c) 2007 INIST/CNRS

File 202:Inspec 1898-1968

(c) 2005 Institution of Electrical Engineers

File 434:SciSearch(R) Cited Ref Sci 1974-1989/Dec

(c) 2006 The Thomson Corp

File 553:Wilson Bus. Abs. 1982-2007/Apr

(c) 2007 The HW Wilson Co

File 9:Business & Industry(R) Jul/1994-2007/Apr 04

(c) 2007 The Gale Group

File 15:ABI/Inform(R) 1971-2007/Apr 04

(c) 2007 ProQuest Info&Learning

File 16:Gale Group PROMT(R) 1990-2007/Apr 04

(c) 2007 The Gale Group

File 20:Dialog Global Reporter 1997-2007/Apr 05

(c) 2007 Dialog

File 148:Gale Group Trade & Industry DB 1976-2007/Mar 27

*Subceptor
Search*

(c) 2007 The Gale Group
File 160: Gale Group PROMT(R) 1972-1989
(c) 1999 The Gale Group
File 275: Gale Group Computer DB(TM) 1983-2007/Apr 03
(c) 2007 The Gale Group
File 476: Financial Times Fulltext 1982-2007/Apr 05
(c) 2007 Financial Times Ltd
File 610: Business Wire 1999-2007/Apr 05
(c) 2007 Business Wire.
File 613: PR Newswire 1999-2007/Apr 05
(c) 2007 PR Newswire Association Inc
File 621: Gale Group New Prod. Annou. (R) 1985-2007/Apr 04
(c) 2007 The Gale Group
File 624: McGraw-Hill Publications 1985-2007/Apr 04
(c) 2007 McGraw-Hill Co. Inc
File 634: San Jose Mercury Jun 1985-2007/Mar 29
(c) 2007 San Jose Mercury News
File 636: Gale Group Newsletter DB(TM) 1987-2007/Apr 04
(c) 2007 The Gale Group
File 810: Business Wire 1986-1999/Feb 28
(c) 1999 Business Wire
File 813: PR Newswire 1987-1999/Apr 30
(c) 1999 PR Newswire Association Inc
File 267: Finance & Banking Newsletters 2007/Apr 02
(c) 2007 Dialog
File 268: Banking Info Source 1981-2007/Mar W4
(c) 2007 ProQuest Info&Learning
File 625: American Banker Publications 1981-2007/Apr 04
(c) 2007 American Banker
File 626: Bond Buyer Full Text 1981-2007/Apr 05
(c) 2007 Bond Buyer
File 95: TEME-Technology & Management 1989-2007/Apr W1
(c) 2007 FIZ TECHNIK
File 98: General Sci Abs 1984-2007/Apr
(c) 2007 The HW Wilson Co.
File 369: New Scientist 1994-2007/Nov W4
(c) 2007 Reed Business Information Ltd.
File 370: Science 1996-1999/Jul W3
(c) 1999 AAAS
File 484: Periodical Abs Plustext 1986-2007/Apr W1
(c) 2007 ProQuest
File 635: Business Dateline(R) 1985-2007/Apr 05
(c) 2007 ProQuest Info&Learning
File 647: CMP Computer Fulltext 1988-2007/Jun W3
(c) 2007 CMP Media, LLC
File 674: Computer News Fulltext 1989-2006/Sep W1
(c) 2006 IDG Communications
File 13: BAMP 2007/Mar W4
(c) 2007 The Gale Group
File 56: Computer and Information Systems Abstracts 1966-2007/Mar
(c) 2007 CSA.
File 75: TGG Management Contents(R) 86-2007/Mar W4
(c) 2007 The Gale Group
File 249: Mgt. & Mktg. Abs. 1976-2007/Mar W4
(c) 2007 Pira International

Set	Items	Description
S1	41	AU='DOUGHTY S'
S2	1	AU='DOUGHTY S G'
S3	6	AU='DOUGHTY S.'
S4	2	AU='DOUGHTY STEPHEN'

S5 4 AU='DOUGHTY STEVEN':AU='DOUGHTY STEVEN G'
S6 3 AU='DOUGHTY, S.'
S7 41 AU='DOUGHTY, S.'
S8 4 AU='DOUGHTY, STEPHEN'
S9 1 AU='DOUGHTY, STEPHEN, 1956-'
S10 9 AU='DOUGHTY, STEVE'
S11 2 AU='DOUGHTY, STEVE, 1973-'
S12 1 AU='DOUGHTY, STEVEN G.'
S13 7 AU='DOUGHTY, STEVEN, G.':AU='DOUGHTY, STEVEN, 1960-'
S14 1 IV='DOUGHTY STEVEN G'
S15 1 IV='DOUGHTY, STEVEN G.'
S16 1 IV='DOUGHTY, STEVEN, G.'
S17 2 AU='BOBBITT C P'
S18 6 AU='BOBBITT CHARLES P':AU='BOBBITT CHARLES P III'
S19 4 AU='BOBBITT, C.'
S20 4 AU='BOBBITT, CHARLES'
S21 2 AU='BOBBITT, CHARLES P.':AU='BOBBITT, CHARLES P. III'
S22 2 AU='BOBBITT, CHARLES, P.':AU='BOBBITT, CHARLES, P., III'
S23 2 IV='BOBBITT CHARLES P':IV='BOBBITT CHARLES P III'
S24 2 IV='BOBBITT, CHARLES P.':IV='BOBBITT, CHARLES P. III'
S25 2 IV='BOBBITT, CHARLES, P.':IV='BOBBITT, CHARLES, P., III'
S26 1719 AU=(BOBBITT(2N) (CHARLES OR CHARLIE OR CHUCK OR CP OR C.P.)
OR DOUGHTY(2N) (STEVEN OR STEPHEN OR STEVE OR SG OR S.G.))
S27 0 BY=(BOBBITT(2N) (CHARLES OR CHARLIE OR CHUCK OR CP OR C.P.)
OR DOUGHTY(2N) (STEVEN OR STEPHEN OR STEVE OR SG OR S.G.))
S28 0 IV=(BOBBITT(2N) (CHARLES OR CHARLIE OR CHUCK OR CP OR C.P.)
OR DOUGHTY(2N) (STEVEN OR STEPHEN OR STEVE OR SG OR S.G.))
S29 1844 S1:S28
S30 8 S29 FROM 347,348,349,350,371
S31 8 IDPAT (sorted in duplicate/non-duplicate order)
S32 5 IDPAT (primary/non-duplicate records only)
S33 1836 S29 NOT S30
S34 99 FINANCIAL() SERVICES() (OFFICE OR OFFICES OR ORGANI?ATION OR
ORGANI?ATIONS) OR FSO OR FSOS OR (CREDIT() CARD OR FINANCIAL OR
INSURANCE) () (INSTITUTION OR INSTITUTIONS OR ISSUER OR ISSUERS
OR COMPANY OR COMPANIES) OR BANK OR BANKS
S35 96 S33 AND S34
S36 2 (KEY OR KEY-) () (VALUE OR VALUES)
S37 0 S33 AND S36
S38 2 DATABASE() (IDENTIFIER OR IDENTIFIERS OR CODE OR CODES OR T-
AG OR TAGS OR ID OR INDICATOR OR INDICATORS)
S39 0 S33 AND S38
S40 1 S35(S) DATABASE?
S41 6 S32 OR S40

*bibliographic
Patent files*

? show files;ds
 File 347:JAPIO Dec 1976-2006/Dec(Updated 070403)
 (c) 2007 JPO & JAPIO
 File 350:Derwent WPIX 1963-2006/UD=200722
 (c) 2007 The Thomson Corporation
 File 371:French Patents 1961-2002/BOPI 200209
 (c) 2002 INPI. All rts. reserv.

Set	Items	Description
S1	74993	FINANCIAL() SERVICES() (OFFICE OR OFFICES OR ORGANI?ATION OR ORGANI?ATIONS) OR FSO OR FSOS OR (CREDIT()CARD OR FINANCIAL OR INSURANCE) () (INSTITUTION OR INSTITUTIONS OR ISSUER OR ISSUERS OR COMPANY OR COMPANIES) OR BANK? ? OR BROKERAGE? ?
S2	7381141	BUILD??? OR CONSTRUCT??? OR CREAT??? OR DEVELOP? OR EXTRAC-T??? OR GENERAT? OR PRODUCE OR PRODUCING OR DERIV? OR SYNTHES-I? OR DISTIL? OR GENERAT???
S3	1499	(KEY OR KEY-) () (VALUE OR VALUES)
S4	681	(KEY OR KEY- OR DATA) () (DEFINITION OR DEFINITIONS OR (ELEM-ENT OR ELEMENTS) () (VALUE OR VALUES))
S5	83	DATABASE() (IDENTIFIER OR IDENTIFIERS OR CODE OR CODES OR T-AG OR TAGS OR ID OR INDICATOR OR INDICATORS)
S6	3105889	MATCH??? OR ALIGN??? OR COMPAR??? OR COMPARATIVE OR COMPAR-I? OR CORRELAT??? OR CORELAT??? OR CORRESPOND???
S7	92895	(FIRST OR 1ST OR SECOND OR 2ND OR THIRD OR 3RD OR THREE OR 3) (2N) (MEMORY OR MEMORIES OR DATABANK OR DATABANKS)
S8	0	S2(5N)S3(5N)S4
S9	3	S3(10N)S6(10N)S7
S10	0	S1(S)S5(S)S8(S)S9
S11	1	S2(10N)S3(10N)S4
S12	1	S2(20N)S3(20N)S4
S13	0	S2(F)S3(F)S4(F)S5(F)S6(F)S7
S14	0	S2 AND S3 AND S4 AND S6 AND S7
S15	4	S9 OR S12
S16	4	IDPAT (sorted in duplicate/non-duplicate order)
S17	4	IDPAT (primary/non-duplicate records only)

full text
Patent files

? show files;ds

File 348:EUROPEAN PATENTS 1978-2007/ 200708

(c) 2007 European Patent Office

File 349:PCT FULLTEXT 1979-2007/UB=20070329UT=20070322

(c) 2007 WIPO/Thomson

Set	Items	Description
S1	73894	FINANCIAL() SERVICES() (OFFICE OR OFFICES OR ORGANI?ATION OR ORGANI?ATIONS) OR FSO OR FSOS OR (CREDIT() CARD OR FINANCIAL OR INSURANCE) () (INSTITUTION OR INSTITUTIONS OR ISSUER OR ISSUERS OR COMPANY OR COMPANIES) OR BANK? ? OR BROKERAGE? ?
S2	1987632	BUILD??? OR CONSTRUCT??? OR CREAT??? OR DEVELOP? OR EXTRAC-T??? OR GENERAT? OR PRODUCE OR PRODUCING OR DERIV? OR SYNTHES-I? OR DISTIL? OR GENERAT???
S3	2995	(KEY OR KEY-) () (VALUE OR VALUES)
S4	1350	(KEY OR KEY- OR DATA) () (DEFINITION OR DEFINITIONS OR (ELEM-ENT OR ELEMENTS) () (VALUE OR VALUES))
S5	692	DATABASE() (IDENTIFIER OR IDENTIFIERS OR CODE OR CODES OR T-AG OR TAGS OR ID OR INDICATOR OR INDICATORS)
S6	1653357	MATCH??? OR ALIGN??? OR COMPAR??? OR COMPARATIVE OR COMPAR-I? OR CORRELAT??? OR CORELAT??? OR CORRESPOND???
S7	52873	(FIRST OR 1ST OR SECOND OR 2ND OR THIRD OR 3RD OR THREE OR 3) (2N) (MEMORY OR MEMORIES OR DATABANK OR DATABANKS)
S8	1	S2 (5N) S3 (5N) S4
S9	8	S3 (10N) S6 (10N) S7
S10	1	S1 (S) S5 (S) S8 (S) S9
S11	0	S1 (F) S5 (F) S8 (F) F9
S12	3	S2 (10N) S3 (10N) S4
S13	1	S9 (F) S12
S14	10	S8 OR S9 OR S10 OR S12 OR S13
S15	4	(S1 (F) S14) OR (S1 (S) S14)
S16	4	IDPAT (sorted in duplicate/non-duplicate order)
S17	4	IDPAT (primary/non-duplicate records only)

*Bibliographic
NPL files*

? show files;ds
 File 2:INSPEC 1898-2007/Mar W4
 (c) 2007 Institution of Electrical Engineers
 File 35:Dissertation Abs Online 1861-2007/Mar
 (c) 2007 ProQuest Info&Learning
 File 65:Inside Conferences 1993-2007/Apr 05
 (c) 2007 BLDSC all rts. reserv.
 File 99:Wilson Appl. Sci & Tech Abs 1983-2007/Mar
 (c) 2007 The HW Wilson Co.
 File 474:New York Times Abs 1969-2007/Apr 04
 (c) 2007 The New York Times
 File 475:Wall Street Journal Abs 1973-2007/Apr 04
 (c) 2007 The New York Times
 File 583:Gale Group Globalbase(TM) 1986-2002/Dec 13
 (c) 2002 The Gale Group
 File 256:TecInfoSource 82-2007/Oct
 (c) 2007 Info.Sources Inc
 File 139:EconLit 1969-2007/Mar
 (c) 2007 American Economic Association
 File 6:NTIS 1964-2007/Apr W1
 (c) 2007 NTIS, Intl Cpyrght All Rights Res
 File 8:Ei Compendex(R) 1884-2007/Mar W4
 (c) 2007 Elsevier Eng. Info. Inc.
 File 34:SciSearch(R) Cited Ref Sci 1990-2007/Apr W1
 (c) 2007 The Thomson Corp
 File 144:Pascal 1973-2007/Mar W4
 (c) 2007 INIST/CNRS
 File 202:Inspec 1898-1968
 (c) 2005 Institution of Electrical Engineers
 File 434:SciSearch(R) Cited Ref Sci 1974-1989/Dec
 (c) 2006 The Thomson Corp
 File 553:Wilson Bus. Abs. 1982-2007/Apr
 (c) 2007 The HW Wilson Co

Set	Items	Description
S1	736507	FINANCIAL() SERVICES() (OFFICE OR OFFICES OR ORGANIZATION OR ORGANIZATIONS) OR FSO OR FSOS OR (CREDIT() CARD OR FINANCIAL OR INSURANCE)() (INSTITUTION OR INSTITUTIONS OR ISSUER OR ISSUERS OR COMPANY OR COMPANIES) OR BANK? ? OR BROKERAGE? ?
S2	21502088	BUILD??? OR CONSTRUCT??? OR CREAT??? OR DEVELOP? OR EXTRACT??? OR GENERAT? OR PRODUCE OR PRODUCING OR DERIV? OR SYNTHESIS? OR DISTIL? OR GENERAT???
S3	823	(KEY OR KEY-) () (VALUE OR VALUES)
S4	1892	(KEY OR KEY- OR DATA) () (DEFINITION OR DEFINITIONS OR (ELEMENT OR ELEMENTS) () (VALUE OR VALUES))
S5	105	DATABASE() (IDENTIFIER OR IDENTIFIERS OR CODE OR CODES OR TAG OR TAGS OR ID OR INDICATOR OR INDICATORS)
S6	11560368	MATCH??? OR ALIGN??? OR COMPAR??? OR COMPARATIVE OR COMPARISON? OR CORRELAT??? OR CORELAT??? OR CORRESPOND???
S7	11533	(FIRST OR 1ST OR SECOND OR 2ND OR THIRD OR 3RD OR THREE OR 3) (2N) (MEMORY OR MEMORIES OR DATABANK OR DATABANKS)
S8	0	S2 (5N) S3 (5N) S4
S9	0	S3 (10N) S6 (10N) S7
S10	0	S1 (S) S5 (S) S8 (S) S9
S11	0	S2 (10N) S3 (10N) S4
S12	0	S2 AND S3 AND S4 AND S6 AND S7
S13	3	S2 AND S3 AND S4
S14	0	S3 AND S6 AND S7
S15	0	S3 AND S7
S16	0	S3 (10N) S4
S17	0	S1 AND S3 AND S4

S18 2 RD S13 (unique items)

Full Text
NPL Files-1

? show files;ds

File 20:Dialog Global Reporter 1997-2007/Apr 05
(c) 2007 Dialog

Set	Items	Description
S1	5207332	FINANCIAL() SERVICES() (OFFICE OR OFFICES OR ORGANI?ATION OR ORGANI?ATIONS) OR FSO OR FSOS OR (CREDIT() CARD OR FINANCIAL OR INSURANCE) () (INSTITUTION OR INSTITUTIONS OR ISSUER OR ISSUERS OR COMPANY OR COMPANIES) OR BANK? ? OR BROKERAGE? ?
S2	16788663	BUILD??? OR CONSTRUCT??? OR CREAT??? OR DEVELOP? OR EXTRAC-T??? OR GENERAT? OR PRODUCE OR PRODUCING OR DERIV? OR SYNTHES-I? OR DISTIL? OR GENERAT???
S3	2333	(KEY OR KEY-) () (VALUE OR VALUES)
S4	693	(KEY OR KEY- OR DATA) () (DEFINITION OR DEFINITIONS OR (ELEM-ENT OR ELEMENTS) () (VALUE OR VALUES))
S5	144	DATABASE() (IDENTIFIER OR IDENTIFIERS OR CODE OR CODES OR T-AG OR TAGS OR ID OR INDICATOR OR INDICATORS)
S6	5804353	MATCH??? OR ALIGN??? OR COMPAR??? OR COMPARATIVE OR COMPAR-I? OR CORRELAT??? OR CORELAT??? OR CORRESPOND???
S7	14852	(FIRST OR 1ST OR SECOND OR 2ND OR THIRD OR 3RD OR THREE OR 3) (2N) (MEMORY OR MEMORIES OR DATABANK OR DATABANKS)
S8	0	S2 (5N) S3 (5N) S4
S9	0	S3 (10N) S6 (10N) S7
S10	0	S1 (S) S5 (S) S8 (S) S9
S11	0	S2 (F) S3 (F) S4
S12	0	S2 (S) S3 (S) S4
S13	0	S3 (10N) S4
S14	0	S3 (F) S4
S15	0	S3 (F) S6 (F) F7
S16	0	S3 (F) S7
S17	0	S3 (S) S7
S18	0	S3 (S) S4

*full text
NPL files-2*

? show files;ds
 File 9:Business & Industry(R) Jul/1994-2007/Apr 04
 (c) 2007 The Gale Group
 File 15:ABI/Inform(R) 1971-2007/Apr 05
 (c) 2007 ProQuest Info&Learning
 File 148:Gale Group Trade & Industry DB 1976-2007/Mar 27
 (c)2007 The Gale Group
 File 275:Gale Group Computer DB(TM) 1983-2007/Apr 03
 (c) 2007 The Gale Group
 File 476:Financial Times Fulltext 1982-2007/Apr 05
 (c) 2007 Financial Times Ltd
 File 16:Gale Group PROMT(R) 1990-2007/Apr 04
 (c) 2007 The Gale Group
 File 160:Gale Group PROMT(R) 1972-1989
 (c) 1999 The Gale Group
 File 621:Gale Group New Prod.Annou.(R) 1985-2007/Apr 04
 (c) 2007 The Gale Group
 File 624:McGraw-Hill Publications 1985-2007/Apr 04
 (c) 2007 McGraw-Hill Co. Inc
 File 634:San Jose Mercury Jun 1985-2007/Mar 30
 (c) 2007 San Jose Mercury News
 File 636:Gale Group Newsletter DB(TM) 1987-2007/Apr 04
 (c) 2007 The Gale Group
 File 13:BAMP 2007/Mar W4
 (c) 2007 The Gale Group
 File 56:Computer and Information Systems Abstracts 1966-2007/Mar
 (c) 2007 CSA.

Set	Items	Description
S1	7706526	FINANCIAL() SERVICES() (OFFICE OR OFFICES OR ORGANI?ATION OR ORGANI?ATIONS) OR FSO OR FSOS OR (CREDIT()CARD OR FINANCIAL OR INSURANCE) () (INSTITUTION OR INSTITUTIONS OR ISSUER OR ISSUERS OR COMPANY OR COMPANIES) OR BANK? ? OR BROKERAGE? ?
S2	26189713	BUILD??? OR CONSTRUCT??? OR CREAT??? OR DEVELOP? OR EXTRAC-T??? OR GENERAT? OR PRODUCE OR PRODUCING OR DERIV? OR SYNTHES-I? OR DISTIL? OR GENERAT???
S3	6223	(KEY OR KEY-) () (VALUE OR VALUES)
S4	6115	(KEY OR KEY- OR DATA) () (DEFINITION OR DEFINITIONS OR (ELEM-ENT OR ELEMENTS) () (VALUE OR VALUES))
S5	1164	DATABASE() (IDENTIFIER OR IDENTIFIERS OR CODE OR CODES OR T-AG OR TAGS OR ID OR INDICATOR OR INDICATORS)
S6	6578863	MATCH??? OR ALIGN??? OR COMPAR??? OR COMPARATIVE OR COMPAR-I? OR CORRELAT??? OR CORELAT??? OR CORRESPOND???
S7	0	S2(5N)S3(5N)S4
S8	1	S2(S)S3(S)S4
S9	3	S3(S)S4
S10	141	S3(10N)S6
S11	0	S10(10N)((FIRST OR 1ST OR SECOND OR 2ND OR THIRD OR 3RD OR THREE OR 3) (2N) (MEMORY OR MEMORIES OR DATABANK OR DATABANKS))
S12	49	S3(F)S4
S13	4	S10(S)S12
S14	0	S1(S)S12
S15	3	S1(F)S12
S16	10	S8 OR S9 OR S13 OR S15
S17	9	S16 NOT PY>1999
S18	9	S17 NOT PD=19991030:20070531
S19	9	RD (unique items)

? show files;ds
 File 610:Business Wire 1999-2007/Apr 05
 (c) 2007 Business Wire.
 File 613:PR Newswire 1999-2007/Apr 05
 (c) 2007 PR Newswire Association Inc
 File 810:Business Wire 1986-1999/Feb 28
 (c) 1999 Business Wire
 File 813:PR Newswire 1987-1999/Apr 30
 (c) 1999 PR Newswire Association Inc
 File 267:Finance & Banking Newsletters 2007/Apr 02
 (c) 2007 Dialog
 File 268:Banking Info Source 1981-2007/Mar W4
 (c) 2007 ProQuest Info&Learning
 File 625:American Banker Publications 1981-2007/Apr 04
 (c) 2007 American Banker
 File 626:Bond Buyer Full Text 1981-2007/Apr 05
 (c) 2007 Bond Buyer
 File 95:TEME-Technology & Management 1989-2007/Apr W1
 (c) 2007 FIZ TECHNIK
 File 98:General Sci Abs 1984-2007/Apr
 (c) 2007 The HW Wilson Co.
 File 369:New Scientist 1994-2007/Nov W4
 (c) 2007 Reed Business Information Ltd.
 File 370:Science 1996-1999/Jul W3
 (c) 1999 AAAS
 File 484:Periodical Abs Plustext 1986-2007/Apr W1
 (c) 2007 ProQuest
 File 635:Business Dateline(R) 1985-2007/Apr 05
 (c) 2007 ProQuest Info&Learning
 File 647:CMP Computer Fulltext 1988-2007/Jun W3
 (c) 2007 CMP Media, LLC
 File 674:Computer News Fulltext 1989-2006/Sep W1
 (c) 2006 IDG Communications
 File 75:TGG Management Contents(R) 86-2007/Mar W4
 (c) 2007 The Gale Group
 File 249:Mgt. & Mktg. Abs. 1976-2007Mar W4
 (c) 2007 Pira International

Set	Items	Description
S1	2363673	FINANCIAL() SERVICES() (OFFICE OR OFFICES OR ORGANI?ATION OR ORGANI?ATIONS) OR FSO OR FSOS OR (CREDIT()CARD OR FINANCIAL OR INSURANCE) () (INSTITUTION OR INSTITUTIONS OR ISSUER OR ISSUERS OR COMPANY OR COMPANIES) OR BANK? ? OR BROKERAGE? ?
S2	9031871	BUILD??? OR CONSTRUCT??? OR CREAT??? OR DEVELOP? OR EXTRAC-T??? OR GENERAT? OR PRODUCE OR PRODUCING OR DERIV? OR SYNTHES-I? OR DISTIL? OR GENERAT???
S3	1980	(KEY OR KEY-) () (VALUE OR VALUES)
S4	1533	(KEY OR KEY- OR DATA) () (DEFINITION OR DEFINITIONS OR (ELEM-ENT OR ELEMENTS) () (VALUE OR VALUES))
S5	264	DATABASE() (IDENTIFIER OR IDENTIFIERS OR CODE OR CODES OR T-AG OR TAGS OR ID OR INDICATOR OR INDICATORS)
S6	2918607	MATCH??? OR ALIGN??? OR COMPAR??? OR COMPARATIVE OR COMPAR-I? OR CORRELAT??? OR CORELAT??? OR CORRESPOND???
S7	12399	(FIRST OR 1ST OR SECOND OR 2ND OR THIRD OR 3RD OR THREE OR 3) (2N) (MEMORY OR MEMORIES OR DATABANK OR DATABANKS)
S8	0	S2 (5N) S3 (5N) S4
S9	0	S3 (10N) S6 (10N) S7
S10	0	S1 (S) S5 (S) S8 (S) S9
S11	0	S2 (10N) S3 (10N) S4
S12	0	S2 (S) S3 (S) S4
S13	0	S3 (10N) S4

*full text
NPL files-3*

S14	1	S3 (F) S4
S15	0	S3 (S) S4
S16	0	S3 (S) S6 (S) S7
S17	0	S3 (S) S7

*IPCA
Search*Research
DatabasesBasic
SearchAdvanced
SearchVisual
SearchSign In | Folder | [Preferences](#) | [New Features](#)
Choose
Databases[Return to the USPTO NP](#)

New Search

Keyword | Publications | Indexes

Find: (financial services organi?ation? or fso? or ((credit card or financial or insurance) and (institution? or issuer? or company or companies)) or bank? or brokerage?) and (build??? or construct??? or creat??? or develop* or extract??? or generate* or produce or producing or deriv* or synthesi* or distil* or generate???) and (key value? or key-value?) and ((key or key-

Search

Clear

??

in Internet and Personal Computing Abstracts

No results were found.

You may want to try your search again after following one or more of these tips:

- Check the spelling of your search terms. Correct any misspellings and re-run the search.
- To broaden your search, use the Boolean operator OR. For example, type: Siamese OR cats.

See [hints](#) for suggestions.[Refine Search](#) [Search History/Alerts](#) [Results](#)

Limit your results:

[Limiters](#) | [Expanders](#)

Date Published Jan Yr: 1980 to Oct Yr: 1999

Peer Reviewed ☐Publication

Expand your search to:

[Limiters](#) | [Expanders](#)Also search for related words ☐Automatically "And" search terms ☐

Search

[Top of Page](#)[EBSCO Support Site](#)[Privacy Policy](#) | [Terms of Use](#) | [Copyright](#)

© 2007 EBSCO Industries, Inc. All rights reserved.

01301385/9 Links

Gale Group Computer DB(TM)

(c) 2007 The Gale Group. All rights reserved.

01301385 **Supplier Number:** 07397312 (This Is The FULL TEXT)

A data base for real-time applications and environments. (HP's Real-Time Data Base) (technical)

Fatchi, Feyzi; Givens, Cynthia; Hong, Le T.; Light, Michael R.; Liu, Ching-Chao; Wright, Michael J.

Hewlett-Packard Journal , v40 , n3 , p6(12)

June , 1989

Document Type: technical

ISSN: 0018-1153

Language: ENGLISH **Record Type:** FULLTEXT; ABSTRACT

Word Count: 8651 **Line Count:** 00665

Abstract: Hewlett-Packard's HP Real-Time Data Base (HP RTDB) is a database management system (DBMS) for developing, managing and interacting with a real-time data base. Real-time databases must store bursts of data quickly and efficiently and provide that information without delay in real-time environments such as factory floor operations. HP RTDB's features include: database definition functions; write and query operations; backup functions; high performance; access to multiple databases; dynamic reconfiguration; security features; and programming aids. The major modules are divisible into user-callable and internal database routines. Details of HP RTDB's data structures, data access, database configuration and design, locking and concurrency control, security, querying and other features are described. HP RTDB runs under HP-UX on an HP 9000 Series 300 or 8000 computer.

Text:

A Data Base for Real-Time Applications and Environments

A REAL-TIME ENVIRONMENT deals with current phenomena rather than past or future events. If information is lost, it is lost forever since there is rarely an opportunity to reclaim it. A typical real-time situation is a factory floor where a computer is monitoring the status of machines and materials and constantly checking to make sure that everything is working properly. Frequently, the data from these checks can be discarded once it is determined that all is indeed satisfactory, although some data might be retained for statistical purposes. If the checking process reveals something amiss, a real-time process might be invoked to correct the situation, such as rejecting a flawed part or shutting down an entire assembly line if a machine is overheating. Data from such incidents is frequently saved for later analysis (e.g., statistical quality control).

A real-time environment needs to respond reliably when an action must be taken quickly within a brief predetermined span of time, such as receiving and storing a satellite data transmission. If the process of receiving and storing the data can always be expected to finish within 80 milliseconds, then the satellite can reasonably transmit every 100 milliseconds without fear of losing any data.

Data capture in a real-time environment may involve sampling large

amounts of raw information with data arriving unexpectedly in bursts of thousands of bytes or even megabyte quantities. A real-time data base must be capable of efficiently storing such large amounts of data and still support the expectations of the user for reliable and predictable response.

When a real-time process requests data, it should be given that data immediately, without any unreasonable delay. Whether or not the data is consistent may be less of a concern than that it is the most current data available. Given sufficient urgency, a real-time application may not require guarantees of either consistency or integrity of data. An application designer must be aware of the risks and should only violate normal data integrity rules when absolutely necessary. A real-time data management system must tolerate such violations when they are clearly intentional.

Finally, a real-time data base must be scalable to the needs of different users. This means that users should be able to implement or eliminate functionality according to the needs of the application. The performance impact of unused functionality must be minimal.

Traditional Data Bases

Traditional data bases are generic and flexible, intended to support the widest possible range of applications. Most traditional data bases use magnetic disc as the primary data storage medium because of its large capacity, relatively high-speed access, and data permanence. Disc-based data bases in the gigabyte range are now possible.

However, traditional data bases are too slow for most real-time applications. Disc access speeds are still two to three orders of magnitude slower than dynamic random access memory (DRAM) access. Even when the average speed of a traditional data base is acceptable, its worst-case speed may be totally unacceptable. A critical need of real-time systems is the ability to provide a predictable response time. Traditional data bases support transaction operations, which may require commit protocols, logging and recovery operations, and access to disc. They also define data access methods that rigidly enforce internal rules of data consistency and integrity. Given a large number of simultaneous transactions, it becomes nearly impossible to guarantee predictable response time. For example, data that is modified as part of an update transaction may not be available to a reader process until the entire transaction is committed. If the reader in this case were a real-time assembly line control process, it could be disastrously delayed waiting for a process of much less importance to complete.

Real-Time Data Bases

Because of extremely high performance requirements, real-time data bases are often custom-designed to the particular needs of a given application. This limits their usability for other applications and causes portability problems if their performance relies upon any hardware characteristics. They are also usually expensive to program and maintain.

Real-time data bases have taken two common approaches, acting either as a very fast cache for disc-based data bases or as a strictly memory-resident system which may periodically post core images to disc. Real-time data bases acting as a high-speed cache are capable of quickly accessing only a small percentage of the total data kept on disc, and the data capacities of purely memory-resident data bases are severely limited by the amount of available real memory. In either case, real-time data bases must coexist with disc-based data bases to provide archival and historical analysis functions in real-time applications. Eventually, a

portion of real-time data is uploaded to a disc-based data base.

Data transfer between real-time and disc-based data bases requires commonly understood data types, and may require reformatting or other treatment to make it digestible to the target data base. Frequently, data is transferred over a network interface as well. The problems of interfacing real-time data bases with disc-based data bases are often further complicated by the customized, nonstandard nature of most real-time data bases.

HP Real-Time Data Base

HP Real-Time Data Base (HP RTDB) is one of the Industrial Precision Tools from HP's Industrial Applications Center. The Industrial Precision Tools are software tools intended to assist computer integrated manufacturing (CIM) application developers by providing standard software solutions for industrial and manufacturing applications problems. The HP Real-Time Data Base is the data base tool. HP RTDB is a set of software routines and interactive query commands for creating and accessing a high-performance real-time data base. It is designed for the specific needs of real-time systems running on HP 9000 Series 300 and 800 Computers.

Fig. 1 shows an overview of the HP Real-Time Data Base system. Access to the data base for the user is through the query commands or an application program written in C that uses the HP RTDB routines. The HP RTDB routines provide the application developer with the ability to:

- * Define or change the data base schema
- * Build the data base in memory
- * Read or write data from or to the data base
- * Back up the schema and data.

The query commands provide an interactive facility for configuring and debugging the data base, and for processing scripts in batch mode and on-line without writing a program. The configuration file is automatically created when the user defines the data base. It contains the system tables and control structures for the data base.

Besides the two interfaces to the data base, HP RTDB also provides the following features:

- * Performance. HP RTDB supports predictable response time, and to ensure speed, HP RTDB is entirely memory-resident. Several design alternatives were chosen to ensure this high performance, such as preallocation of all data base memory to minimize memory management overhead, alignment of data on machine word boundaries, simple data structures, and the extensive use of in-line macros to reduce the overhead of function calls. The design alternatives chosen produced performance results that exceed initial goals. For example, performance tests showed that 66,666 56-byte records can be directly retrieved from an HP Real-Time data base in one second.

- * Multiple Data Base Access. HP RTDB resides in shared memory so that multiple processes can access the data base concurrently. Also, one process can access multiple data bases.

- * Simple Data Structures. Data is stored in HP RTDB in two forms: tables and input areas. A table is an array of columns and rows (tuples) that contain related information (see Fig. 2). Input areas are areas in the data base designed to receive large blocks of unstructured data. Often this data comes from high-speed data acquisition devices.

- * Data Access. Retrieval routines are constructed so that any specified data can be accessed directly, sequentially, or by hash key values. Direct access is available to a tuple (row) of a table and to an

offset in an input area.

- * Dynamic Reconfiguration. Tables and input areas can be added or deleted quickly without having to recreate the data base.

- * Security. HP RTDB provides three levels of password protection: data base administrator access, read-write access, and read-only access.

- * Backup and Recovery. The schema (data base structure), and optionally the user's entire data base can be saved to a disc file.

- * Locking. HP RTDB provides tables and input area locking. This means that an application can exclusively access a table or input area until it decides to release the lock. If the application requires, read-through and/or write-through locks are allowed.

- * Scalability. HP RTDB is scalable. If some features are not required they can be eliminated to improve performance.

- * Documentation Aids. HP RTDB is supplied with a self-paced tutorial complete with a query/debug script to build the data base that is used in the tutorial examples. There is also an on-line help facility for the interactive query/debug utility.

- * Programming Aids. HP RTDB programming aids include:

- * A standard C header file defining the constants and data structures required to use the HP RTDB subroutines

- * Prototyping and debugging capabilities of the query/debug utility

- * On-line access to explanations of HP RTDB error codes

- * User-configurable error messages, formats, and hooks for user-written error routines

- * Native language support which includes 8-bit data, 8-bit filenames, and message catalogs.

HP RTDB Modules

The HP Real Time Data Base modules can be grouped into two main categories: user-callable routines and internal data base routines (see Fig. 3). The user-callable routines include the following functions.

- * Administrative Functions

- * Define the data base including its name, passwords, and system limits (MdDefDB)

- * Build or rebuild the data base in memory (MdBuildDb)

- * Remove a data base from memory (MdRmDb)

- * Change data base system limits or passwords (MdChgDb, MdChgPwd).

- * Data Definition Functions

- * Define a table or input area (MdDefTbl, MdDefIA)

- * Define or add column(s) to a user table (MdDefCol)

- * Define an index on column(s) in a defined table (MdDefIdx)

- * Remove a table or an input area (MdRmTbl, MdRmIA)

- * Remove an index from a table (MdRmIdx).

- * Session Begin or End Functions

- * Open the data base and initiate a session (MdOpenDb)

- * Close the data base and terminate a session (MdCloseDb).

- * Data Manipulation Functions

- * Open a table or input area for access (MdOpenTbl, MdOpenIA)

- * Get a tuple by sequential search (MdGetTplSeq), hash key index (MdGetTplIdx), or tuple identifier (MdGetTplDir)

- * Compare a tuple value with a set of expressions (MdCompare)

- * Add or remove a tuple to or from a table (MdPutTpl, MdRmTpl)

- * Update a tuple (MdUpdTpl)

- * Get or put a value from or to an input area (MdGetIA, MdPutIA)

- * Lock or unlock a table or an input area (MdLock, MdUnlock).

- * Utility Functions
- * Save the data base schema and optionally the entire data base to disc (MdTakeImage)
- * Release resources held by prematurely terminated processes (MdCleanup)
- * Provide information on the columns of a table (MdColInfo)
- * Provide information on the minimum data base and schema size in bytes (MdDbSizeInfo)
- * Provide information on all or one specific user table, index on a table, or input area (MdTblInfo, MdIxInfo, MdIAInfo)
- * Provide information on system tables and session use (MdSchInfo).

The internal data base routines are used by either the user-callable routines or other internal routines. They are implemented as C functions or macros. The macro implementations are used for small pieces of code. The resulting code is slightly larger but faster. The functions performed by the internal data base routines include:

- * System Table Manager. These routines handle the tables that define the schema and configuration of the data base.
- * Index manager. These routines handle hashing, index manipulation, and formulation of hash index key values.
- * Concurrency Manager. These routines handle locking operations and control concurrent processes using the data base.
- * Storage Manager. These routines handle memory management, which includes keeping track of allocated and available shared memory.
- * Operating System Interface. These routines provide a clean and consistent interface to the HP-UX operating system.
- * Table and Tuple Manager. These routines handle functions related to tuples and tables such as copying, adding, or deleting tuple values.

Data Structures

The data structures in HP RTDB are divided into two categories: those that manage and control access to the data base and define the schema, and those that contain the user data. Fig. 4 shows an overview of these

structures in shared memory. The data structures in the schema and control section are automatically created when the data base is defined. The data structures in the user area are added later when the data base is built. Only two of these data structures are visible and accessible to the user--user tables and input areas.

- * Main Control Block. The main control block contains the data base status, limits, and pointers to other data structures in the schema and control section of the data base. It also contains information used by the storage manager, such as pointers to the beginning and end of free memory storage space, a pointer to the list of free storage blocks, and the total amount of free storage left.

- * Session Control Blocks. A session control block is allocated to each process accessing the data base. Each block contains a session identifier, a pointer to the main control block, and other information about the process, such as the user identifier (HP-UX uid) and the process identifier (HP-UX pid). The session identifier is returned to the user when the data base is opened, and is used in subsequent calls to access the data base. The number of session blocks determines the number of users that can have access to the same data base at any one time. This number is determined when the data base is created.

- * Semaphore Control Blocks. There is a semaphore control block for

each lockable object in the data base (i.e., user tables and input areas). These blocks contain HP-UX semaphore identifiers.

- * Locks-Held Table. Each entry in the locks-held table indicates whether a lock is being held by a session on a certain data base object (user table or input area), and if so, what type of lock.

- * Index Tables. Index tables contain the data for performing fast access (i.e., hash indexing) to system and user tables.

- * System Tables. System tables contain the schema (structure) of the data base and information about the locations and attributes of all data base objects, including themselves.

- * User Tables and Input Areas. The application data managed by the user is contained in the user tables and input areas.

Tables. The table, which is a two-dimensional array consisting of rows (tuples) and columns, is the fundamental data structure in HP RTDB. There are three types of tables: system tables, user tables, and index tables. All tables, whether they are system, index, or user tables, have the same structure, called a table block (see Fig. 5). A table block is divided into two sections: control structures and data. Control structures contain the information needed to locate, add, or delete data in a table. The data portion of the table contains the system or user data. The information in the control structures includes:

- * Table Block Header. The header contains information needed to access information within the table, such as data offsets and table type (i.e., system, index, or user).

- * Slot Array. Each entry in the slot array indicates whether a tuple in a table is filled or vacant. The slot array is accessed when adding or deleting tuples, and when searching sequentially.

- * Column Descriptor Array. The entries in the column descriptor array describe the columns in the data portion of the table block. Each column descriptor defines the column type (i.e., character, byte string, integer, float, input area offset, etc.), the column length, and the column offset in bytes from the start of the tuple (see Fig. 6).

The data in each type of table is stored in tuples. The tuple format, which is the number, length, and type of columns, must be the same for all tuples in any one table. However, the tuple format may be different for each table. The number and size of tuples in a table are limited only by the amount of real memory available. Each tuple and all columns within a tuple are word-aligned. Variable-length columns and null columns are not supported. To support only fixed-length data and columns may seem wasteful of real memory, but this scheme more than offsets the increased size and complexity of code needed to support variable-length data, and the resulting performance degradation. Another benefit is that the size of a table has little effect upon the speed of accessing any given tuple. Since all tuples in a table are the same length, a tuple's location is fixed and can be quickly determined with one simple calculation. Once located, copying the tuple's data between the data base and a user buffer can be done by words (four bytes at a time) rather than one byte at a time, since all data is aligned on machine word boundaries.

Data in user tables can be any supported C data type or an offset into an input area. Users can also store and retrieve unsupported data types in table columns defined as a byte string type. Using the byte string type, the user can store pointers to other tuples in the same or any other table. Data compression, alignment of values in tuples, and verification of data types is left to the user's application, where these functions can be

done more efficiently. HP RTDB routines store user data exactly as it is received and retrieve user data exactly as it is stored. A positive side effect of this is that the storage and retrieval integrity of 16-bit data (e.g., Katakana or Chinese text) can be guaranteed without special routines.

Because all table types have the same table block structure, the same code can be used to perform operations on system, index, and user tables. However, system table access is so critical to performance that operations on system tables are often performed by special code that takes full advantage of the known, fixed locations and formats of system tables.

Tuple Identifiers. A tuple identifier or tid uniquely identifies each tuple in every table in the data base including system tables, index tables, and user tables. Tuple identifiers are used by the user to access user tables and by internal HP RTDB routines to access all the tables in the data base. A tuple identifier is returned to the user when a tuple is added to a table (MdPutTpl) or after a successful table search (MdGetTplSeq) or after a successful indexed access (MdGetTplIx). Once obtained, a tuple identifier can be used in subsequent calls to provide extremely fast, direct access to the same tuple for rereading, deletion, or update. Directed access by tuple identifier is by far the fastest access

ess

method in the HP RTDB data base.

The data type for a tuple identifier is called tidtype and contains three elements: a table number, a tuple number, and a version number.

* The table number is the tuple number for a tuple in a system table that describes the table associated with the tuple identifier. Fig. 7 shows the tid for a user table and the use of the table number and tuple number entries. For system and user tables, the system table containing the tuples of table descriptions is called a table system table, and for index tables the system table is called an index system table. System tables are described in detail later in this article.

* The tuple number indicates the row in a table containing the tuple data.

* The version number is used to ensure that a tuple being accessed directly by a tid is the same tuple that was accessed when the tid was first obtained. For example, suppose user A adds a tuple to table X and saves the returned tid for subsequent rereading. If user B accesses table X and deletes the tuple added by user A and then adds another tuple to table X, it is possible that the tuple added by user B could occupy the same location as the tuple originally added by user A. When user A attempts to use the same tid on table X for reading the tuple that was changed by user B, the version numbers won't match and user A will be prevented from accessing the tuple and notified of the problem.

Tuple identifiers can be used to chain tuples logically. Users can build logical relationships between tuples by inserting the tuple identifier of one tuple as a column value of another tuple. This concept is illustrated in Fig. 8, where tidA4 and tidB2 are tuple identifiers. The tuple identifier is designed so that its value remains constant across data base restarts and dynamic schema changes. Thus relationships of tuples, whether system-defined or user-defined, are not lost when a data base is shut down and restarted.

System and User Tables. The system tables contain the schema for the data base, and the user tables and input areas contain the user's

application data. The relationship between these data structures is shown in Fig. 9. There are four types of system tables:

- * Table System Table. The table system table contains information on all the user tables and system tables in the data base including itself. One section of the table describes system tables and another section describes user tables. Each tuple in the table system table describes one table, and the columns contain relevant information about the attributes of the table described by the tuple (e.g., table name, tuple length, number of columns, and so on). Fig. 10 shows a portion of a tuple in the table system table for a user table (UserTbl02). The entry CSTtid is the tuple identifier for the starting tuple in the column system table assigned to UserTbl02, and the entry ISTtid is the tuple identifier for the starting tuple in the index system table assigned to UserTbl02. The entry FstBlkOff is an offset in bytes to the first block of storage for UserTbl02. When the user adds or deletes a table, the table system table is updated accordingly. Likewise, when certain changes (e.g., add indexes) are made to the user table these changes are reflected in the associated tuple in the table system table.

- * Column System Table. The column system table contains information on all columns in a user table. Each tuple describes one column in a user table. Some of the information kept in the column system table includes column type, length, and offset for each user table column. This same information is kept in the column descriptor array of the user table control block described earlier. The reason for having this data in two places is that it eliminates one level of indirection when accessing data in user table columns. A new tuple is added to the column system table when a new column is added to a user table.

- * Index System Table. The index system table contains information on the indexes for system and user tables. Each tuple in the index system table describes an index defined on a system or user table. Indexes on system tables are predefined by HP RTDB and indexes on user tables are defined only by the user. Indexes are described in more detail later in this article.

- * Input Area System Table. The input area system table contains information on user-defined input areas. Each tuple contains the input area name, the input area size, and the offset (in bytes) of the beginning storage location allocated to the input area.

Indexes. Indexes are defined on tables to provide faster access to a data tuple. HP RTDB provides hash indexing. Fig. 11 shows the organization of the hash indexing scheme employed in HP RTDB. In this scheme a key value, which is composed of one or more columns in a table, is sent to a hash function that computes a pointer into a table of tuple identifiers. Once the tuple identifier is known, the desired tuple can be accessed.

The columns that are used for the key value are designated in the index system table described earlier. Fig. 12 shows the relationship between the index system table and the columns in a user table designated for key values. These columns are specified when an index is defined for a table. In many hashing schemes the hashing function transforms a key value into a storage address where the user's data is stored. HP RTDB does not use hashing for both storage and retrieval of tuples, but only as a very fast retrieval mechanism.

The process of inserting a new tuple into a table with a hash index takes the following steps:

- * The tuple is inserted in the first available slot in the user table

without regard to any index defined on the table.

- * A location is found in the index table by applying the hash function to the key value of the tuple. This location is called the primary location for the tuple. If the hash function returns a primary location that is already in use, a secondary location is found and linked to the primary location using a synonym chain.

- * The tuple identifier of the inserted tuple is stored in the designated primary (or secondary) location in the index table.

The process of retrieving an existing tuple from a table using the tuple's key value takes the following steps:

- * The hash function is applied to the key value to obtain the primary location for the corresponding tuple identifier in the index table.

- * If the primary location has no synonyms, the tuple addressed by the tuple identifier in the primary location is accessed and returned.

- * If synonyms exist, then each one is accessed in turn until one is found with a key value that matches the unhashed key value of the requested tuple. If the hash index is defined with the option to allow duplicate key values, then each tuple with a matching key value will be returned in the order found.

Independence of retrieval from storage provides HP RTDB with some major advantages:

- * Multiple hash indexes. Each table can have multiple hash indexes defined for it, allowing the same table to be searched with any number of different keys as shown in Fig. 12.

- * Constant tuple identifiers. A hash index can be rehashed without causing any data migration (the data tuple locations do not change). This means that applications can use direct access by tuple identifier and never be concerned that rehashing might cause a tuple identifier to change. This feature also significantly improves the performance of applications that frequently update table columns used for key values.

- * Dynamic hash index definition. Unlike direct hashing algorithms, hash indexes can be added to or removed from existing tables.

- * Fixed space overhead. The space overhead incurred because of defining a hash index is a direct function of the number of tuples in a table and does not depend on the number of columns, so it does not increase as new columns are added to a table.

However, no matter how carefully a hash function is designed, it cannot guarantee that collisions will not occur when the function is applied to diverse and unpredictable sets of keys. Therefore, when a collision does occur, there must be a means of specifying an alternate location in the index table where the new tuple identifier can be stored. HP RTDB uses a form of separate chaining in which each hash index consists

of two segments, a primary segment and a secondary segment (see Fig. 13).

The primary segment contains the tuple identifiers of all keys that hash to an unused primary location. To reduce the probability of collisions, the number of locations in the primary segment can be configured by the user to be more than the maximum number of tuples in the data table. For example, if the number of primary segment locations is 1.25 times the number of tuples in the data table (primary ratio), then the load factor (packing density) of each primary segment cannot exceed 80 percent. When this means is that for a table with eight tuples the number of primary segments is 10 (1.25×8), and if there are no collisions, at most eight of the tuples in the primary segment will be occupied. A higher primary ratio

will reduce the probability of collisions but will increase the primary segment size. Users can adjust each index's primary ratio to achieve the best performance and minimum memory consumption.

The secondary segment contains the tuple identifiers of all data values the hash to a primary location that is already in use. This segment provides a separate overflow area for secondary entries (synonyms), thus eliminating the problem of migrating secondaries (existing synonyms that must be moved to make room for a new primary entry). The secondary segment is allocated based upon the number of tuples in the data table and is guaranteed to be large enough for even a worst-case index distribution. After a collision occurs at a location in the primary segment, the primary location becomes the head of a linked-list synonym chain for all secondaries that hash to the primary location.

Input Areas. Input data in a real-time environment may be expected or unsolicited, and can arrive in streams, small packets, or large bursts. This data may also involve complex synchronization of processes to handle the data. In all cases, there is a need to respond to the arrival of the new data within a predictable time before it is too old to be of value, or is overwritten by the next arrival.

Input areas provide highly efficient buffering for receiving and storing unstructured data into the data base. Users can configure a data base with any number of input areas of any size up to the limits of available shared memory. Values in input areas can be read or updated either using offsets in a named input area or, for even higher performance, using an input area's actual address as if it were a local array variable. Like tables, input areas can be explicitly locked and unlocked for control of concurrent access.

Data Access

Traditional data base transactions are not supported in HP RTDB because each access to the data base is considered a transaction, and each access is guaranteed to be serialized and atomic. However, a system designer can still define and implement an application transaction as a set of two or more data base accesses, which will complete or fail as a group.

The general data access flow in HP RTDB is shown in Fig. 14. The sequence of events to access the data base to update a tuple would be:

- * Obtain the address of the main control block using the session identifier (SessID). The session identifier is returned to the user when the data base is opened and is used in subsequent calls to access the data base.

- * Obtain the address of the table system table from the main control block, and using the table identifier (TblTid) obtain the tuple of the user table from the table system table. The user is given a table identifier when the table is opened.

- * Obtain the entries in the locks-held and semaphore control blocks and lock the user table. The addresses for these entries are obtained from the main control block.

- * Finally, obtain access to the tuple in the user table using the user table address obtained from the table system table and the tuple identifier (tid).

This process is the same for input areas, except that the input area system table is accessed rather than the table system table, and the input area offset is used instead of the tuple identifier.

Performance tests to assess the data access performance characteristics of the HP Real-Time Data Base routines were run on an HP

9000 Model 825. The benchmark for these tests consisted of 56-byte tuples. During the tests, shared memory was locked into physical memory. The results of these performance tests are summarized in Fig. 15.

Table Access. There are three methods of locating tuples in user tables. Tuples can be read using sequential access, hashed key index access, or direct tuple identifier access. However, updates and deletions of tuples can only be done by direct tuple identifier access. This means that to update or delete a tuple, it must first be located by one of the three read access methods. Sequential access starts at the beginning of a table and returns each tuple in the order in which it is encountered. Since tuples can be inserted or deleted at any location, the physical order of tuples in an active table is unpredictable. The user can set up search conditions for sequential searching, such as the number of comparisons, columns to use, comparison operator (e.g., EQ, NE, etc.), and ASCII type (e.g., hexadecimal, octal, etc.). If the search conditions are specified, then only those tuples that meet the conditions are returned. Sequential access is the slowest mode of access, but for very small tables of 10 to 15 tuples, the speed of sequential searching is comparable with indexed searching. Sequential access is most appropriate for serially processing most or all of a table's data, since it does not use additional memory for index tables.

Indexed access, which uses the indirect hashing technique discussed earlier, is much faster than sequential access, but still slower than direct access by tuple identifiers. Index keys can consist of up to five contiguous or noncontiguous table columns of mixed data types and any number of indexes can be defined for a single table. Although there is no hard limit as to how many indexes can be defined for a table, each index requires additional memory for index tables and additional processing time to create or update each index key defined. Indexed access is best for applications that need fast, unordered access to data and that mainly perform reads and updates rather than insertions and deletions.

The HP RTDB tuple update routine allows three alternative courses of action when columns that make up an index key value are about to be updated. One option specifies that the indexes should be modified (that is, rehashed) to reflect the update. A second option is to deny the update and return an error when an index key value is about to be changed. For top performance, there is an option to update a tuple and bypass checking for index modification. This option should only be used if the user's application can ensure that a tuple's index key values are never changed after its initial insertion into a table.

Direct access by tuple identifier is by far the fastest form of access. A tuple's tuple identifier is returned when it is first added to a table and also when the tuple is accessed by a sequential or indexed search. The returned tuple identifier can then be used to update, delete, or reread the same tuple directly any number of times since tuple identifiers do not change, even when a table is rehashed. This feature offers spectacular benefits when a small set of tuples is repeatedly accessed. The tuple identifier can be obtained once, stored internally, and then used to access the same tuples directly in all subsequent operations.

Input Area Access. Data in input areas can only be read or updated. Since input areas are usually updated by being overwritten with new input data, HP RTDB does not provide separate routines for deleting and inserting data elements in input areas. Nor are these functions really needed, since updating an element to a null or non-null value accomplishes the same end.

Since the structure and content of input areas are application dependent and can change at any time, HP RTDB does not try to map input areas as it does tables. Data elements in input areas are accessed by naming the input area and specifying the element's offset and length, HP RTDB then locates the start of the input area, adds the offset, and reads or updates the selected data element. For maximum performance, the input area may optionally be addressed directly as if it were a local array, but this access mode bypasses HP RTDB's address validity checking and concurrency control mechanisms and should only be used if the extra performance is critical. The application must then ensure correct addressing to avoid data base corruption.

Users can also associate a table column with an input area data element by defining the column's data type as a pointer to an input area and then assigning the data element's offset as the column's value. The input area offsets shown in Fig. 8 are input area pointer types.

Configuration and Creation

Much effort was made to keep the process of configuring and creating a data base as simple and flexible as possible. The final definition of data base objects intended for future implementation can be deferred until they are actually needed, and other data base objects can be added and/or removed after the data base is created. Also, all data base configuration and maintenance functions can be done with the interactive HP RTDB query/debug commands as well as by calling HP RTDB subroutines. This allows users to prototype and test data base designs without writing a single line of program code.

Defining the Data Base Schema. The first step in creating an HP RTDB data base is to define the system limits of the data base, that is, the data base name, the configuration file name, the maximum number of tables, input areas, indexes, columns, and sessions that will be accommodated by the data base at any time. The user may choose to defer the actual definition of some data base objects until a later time, but must inform HP RTDB of how many of each object may eventually be defined. This information is used to ensure that the system tables and control structures for the data base are as large as the maximum requested. Preallocation of continuous storage for the maximum size of the data base objects instead of allocating on an as-needed basis eliminates the overhead of checking for available space when adding tuples to a table. It also eliminates the overhead associated with dynamically allocating and deallocating blocks in a multiuser environment.

When the system limits are defined, a sketched schema and control structures are generated in shared memory and saved on disc in the configuration file. At this point the data base schema can be filled out with the definition of user tables, columns, input areas, and indexes either through query/debug commands or by calls to the HP RTDB subroutines (MdDefTbl, MdDefCol, MdDeflx, and MdDeflA). As these other data base objects are defined, the information about them is entered into the system tables and the schema grows more complex. However, no storage is allocated for newly defined data base objects until the data base is built. The user can at any time save a copy of the current memory-resident schema to the
co

nfiguration file. When the data base is fully operational and contains data, the data as well as the schema can be saved in the same file.

Building a Data Base in Memory. Once the system limits of the data

base are set and the schema defined, the data base can be built. First, the schema must be loaded into memory. The schema will already be in memory if the data base is newly defined. Otherwise, the schema file on disc is opened and loaded into memory (MdOpenDb). Using the memory-resident schema data, the HP RTDB build routine (MdBuildDb) allocates shared memory to each data base object builds and initializes any data or control structures associated with the objects, and sets up the logical links between the structures. HP RTDB also provides routines to calculate the minimum memory needed to build the data base from the schema. Additional memory may optionally be allocated to allow for future implementation of data base objects that are not yet defined in the schema. After a data base is built, it is ready to be initialized with application data.

Locking and Concurrency Control

There are three components to the synchronization of concurrent access to a data base: session management, lock management, and semaphore management. As each new user process opens the data base, HP RTDB allocates a session control block for the new process and the process becomes attached to the data base. A session identifier is returned to the process for use in subsequent calls to access the data base, and the session control block is filled with information about the process such as the HP-UX process identifier (pid) and user identifier (uid). The session identifier is used to index into the locks-held table. With this and other data the session manager is able to perform its role in controlling concurrent access to the data base.

Locking in HP RTDB is provided only at the table and input area level rather than at the tuple and data item level. This coarse granularity of locking is acceptable because in a memory-resident data base, locks are normally held for very short periods of time. Each object (user table and input area) in the data base is owned by a specific semaphore. Locking of data base objects is accomplished by acquiring the object's semaphore and associating it with the process's session identifier. The lock is released by freeing the semaphore and breaking the link between the semaphore and the session identifier.

HP RTDB controls the locking and unlocking of semaphores, but all queuing and rescheduling of blocked processes is handled by the HP-UX operating system. This gives HP RTDB a simple, efficient, and reliable concurrency control mechanism that is guaranteed to be compatible with other HP-UX features. For example, a user could easily integrate HP RTDB and the HP real-time extension features to implement real-time priorities in an application. HP real-time extensions are additions to HP-UX that allow users to set high priorities for certain processes.

If the application does not explicitly lock a data base object before trying to access it, the HP RTDB routine called to do the access will normally apply an implicit lock of the object by default. There are options to allow users to read and write through locks, but these options may compromise the integrity of the data base and should be used with caution when higher performance is critical. A read-through lock allows a high-priority process to access data in the data base that may be in the process of being updated by another process.

Security

HP RTDB provides three levels of security through the use of passwords. A password is required to access the data base. The password level, which can be data base administrator, read-write, or read-only, determines the user's access authorization. The data base administrator

password allows a user process to perform any operation on the data base supported by HP RTDB subroutines or by the query/debug commands. the read-write password allows a user process to read, modify, and delete data in the data base, but not to perform any of the data base definition functions, such as adding or removing a table or index. The read-only password allows a user only to read data in the data base, but not to delete or modify data, or perform any data base definition functions.

In addition to the password security, the data base administrator (or the root user) can also alter the access permissions of the schema file on disc or the data base's shared memory segment to limit access to the data base.

Backup and Recovery

Memory-resident data bases are particularly vulnerable to power failures and operating system failures because both types of failures usually destroy the contents of main memory. Battery backup power systems can provide excellent protection against power failures, but system failures pose a problem that really has no good solution.

The traditional backup methodology of logging all changes to a disc file cannot be used if high performance is desired; yet there is no other way to keep a secure backup with close parallel to the state of memory. HP RTDB provides a "snapshot" backup which allows each application to choose an acceptable trade-off between performance and secure backup.

At any time, the application can call an HP RTDB routine to save an image of the data base schema or the schema and data to a disc file. For a data base of 34,000 bytes consisting of 6 tables and 2 input areas, a single snapshot takes about 0.5 second on an HP 9000 Series 825. Snapshots can be taken as often or as rarely as the user application chooses, and can be triggered periodically or by specific events. Users who can afford the overhead can take more frequent backups while users who require top performance may rarely or never take a backup. In some real-time applications, there is little point in taking a backup since the data would be obsolete long before the system could be restarted. Data base recovery is also very fast but a data base can only be recovered to the point where the last snapshot was taken. either the schema or the schema and the data can be recovered. Recovery of the schema only would create an empty data base which could then be reinitialized with data by the application.

Query/Debug Utility

The query/debug utility is included as part of the HP Real-Time Data Base software to provide real-time application developers with a tool to:

- * Assist with prototyping, testing, and debugging applications
- * Create, modify, and maintain HP RTDB data bases in both development and production environments
- * Use as a simple and flexible HP-UX filter, which when combined with other HP-UX commands and utilities, can provide useful application functions to HP RTDB users without the need for additional code.

The query/debug utility supports nearly all of the functionality of the HP RTDB subroutines. However, it is highly generalized and is designed to be safe and friendly rather than fast. Therefore, most query/debug functions are significantly slower to execute than equivalent subroutine calls.

The query/debug command syntax is modelled after the Structured Query Language (SQL), an ANSI industry-standard relational interface. This resemblance to SQL is intended only to make it easier for users who are already familiar with SQL. The query/debug utility is not intended to

support the SQL standards of inquiry or reporting functionality.

The query/debug utility was designed as an HP-UX filter and conforms to the conventions for filters in its use of HP-UX input/output files stdin, stdout, and stderr. This allows it to be used with input and output redirection, pipes, and so on. Output can optionally be produced without headings to enable clean output data to be piped directly into other filters or user-written programs.

Query/debug commands can be entered interactively for ad hoc work, or can be read from ordinary disc files for repetitive tasks. For example, the commands to define and initialize a data base could be saved in a disc file to ensure that the data base is always recreated the same way. Likewise, simple reports can be generated using query/debug command files or by combining query/debug command files with HP-UX shell scripts and utilities.

The query/debug commands provide the following functionality:

- * Define, reconfigure, build, remove, and back up a data base
- * Change passwords and shared memory security permissions
- * Initialize table and input area values in a new data base
- * Display data base configuration and status information
- * Generic add delete, update, and select of tuple values based upon indexed or sequential searches
- * Display or print all or selected data from all or selected tuples in a table in either tabular or list format
- * Generic update, select, and display of input area values
- * Load or store data base values from or to disc files
- * Debugging aids such as hexadecimal data display, a "peek" function, and an error trace option for tracing all errors that may occur during any query/debug processing of the user's data base
- * On-line help facility for all query/debug commands
- * Built-in octal, decimal, and hexadecimal integer calculator
- * Execution of HP-UX commands without leaving query/debug.

Conclusion

The goal of a high-performance data base drove many of the design decision and implementation techniques for HP Real-Time Data Base. The performance goals were met and exceeded with simple data structures (tables and input areas), programming techniques such as macros, and options that allow users to eliminate features that affect performance. The result is a set of routines that enables real-time application developers to create custom data bases for capturing and retrieving the diverse data structures found in real-time environments.

Acknowledgments

The authors wish to thank Marie-Anne Neimat, Ming-Chien Shan, and Bob Price for their assistance in the design of RTDB, and Mark Butler for his direction in the creation of RTDB.

Captions:

An overview of the HP Real-Time Data Base System. (chart); A table structure in HP RTDB consisting of six tuples and five columns. (table); HP RTDB module hierarchy. (chart)

COPYRIGHT 1989 Hewlett Packard Company

Special Features: illustration; chart; table

Company Names: Hewlett-Packard Co.--Products

Descriptors: DBMS; Specifications; System Design; Real-Time System; Applications; Software Packages

SIC Codes: 7372 Prepackaged software

Ticker Symbols: HWP

Trade Names: Real-Time Data Base (Data base management system)--Design and construction

File Segment: CD File 275